****

**Student ID: C00131026**

**Student Name: Guanting Su**

**Supervisor: Joseph Kehoe**

**Date of submission: 17.1.2011**

**Design Manual**

**<STM Lua>**

**(CW228)**

Content

[1. Introduction 1](#_Toc286082868)

[2. Low Level Design 2](#_Toc286082869)

[2.1. Data Design 2](#_Toc286082872)

[2.2. Architectural Design 3](#_Toc286082873)

[1) Create Transaction 3](#_Toc286082874)

[2) Get valuables to create a table 4](#_Toc286082875)

[3) Start Transaction 4](#_Toc286082876)

[4) Read 4](#_Toc286082877)

[5) Write 4](#_Toc286082878)

[6) Commit transaction 4](#_Toc286082879)

[2.3. Procedural Design 5](#_Toc286082880)

[2.3.1. Algorithm 5](#_Toc286082881)

[3. High Level Design 7](#_Toc286082882)

[3.1. Use case diagram 7](#_Toc286082886)

[3.2. Use cases 7](#_Toc286082887)

[3.3. System sequence diagram 9](#_Toc286082888)

[4. Application Testing Scheme 9](#_Toc286082889)

[5. Conclusion 10](#_Toc286082890)

[6. Reference 11](#_Toc286082891)

# Introduction

This document is introduced the design of developing a library of Lua for software transaction memory (STM) control. STM is new technique to control concurrency program. Others developers can achieve concurrency with my own library. In this library, it supports:

* Create a new transaction
* Add valuable to transaction
* Start a transaction
* Commit transaction
	+ - Check version
		- Update

At last it can achieve running one or multi transactions by one or multi threads. And the target groups of this project are developers who want to develop concurrency programming in Lua. Because this project is developed into a library of Lua, so the developers if they need the library, they just need use require to import the library. And then the developers can invoke the functions that are from the library. Below is example:

|  |  |
| --- | --- |
| User Code | Compiled Code |
| int foo(int arg){…atomic{ b=a+5}…} | int foo(int arg){ jmpbuf … do{ if(setjmp(env)==0){ stmStart() temp=stmRead(a) temp=temp+5 stmWrite(b, temp1) stmCommit(); break; } }while(1) …} |

Figure 1 STM example [1]

# Low Level Design

1.
2.

## Data Design

This part will focus on describe the data structure is used in Software Transaction Memory (STM) Lua and describe the relationship between data structure. In Lua, save value in a new object that is created by my library. The valuable is saved as table by a function, and the value and version of valuable in the table. If there are multi valuables, users can use array save the valuables. For example, array[] value. The below is showing an example:



Figure 2 the data structure of table

Before the transaction start, the value of valuable will save as a copy value.

 table b

 value= transaction.read(b[1][1])

 versionNo= transaction.read(b[2])

newValue=value.copy()

When the commit transaction checks the object whether is available, and the version No. whether is match. If the version is match, it will update the copy value as the new value of object. If not match, it means the object is used by others before. No matter the update is successful or not, the transaction will unlock the object.

 currentVersionNo= transaction.read(b[1])

If (versionNo== currentVersionNo)

 Transaction.write(object[0], newValue)

 Transaction.write(object[1], versionNo++)



Figure 3 update the version number

## Architectural Design

This project will be developed into a library. The library will support create transaction, transaction begin, create a table to save valuable, get value of valuable, commit to check lock and version and then update new value to valuable.



Figure 4 showing the overview of application architecture

### Create Transaction

Create a new transaction to store the users need process data, because a transaction runs in isolation, meaning it executes as if it’s the only operation running on the system and as if all other threads suspended while it runs. Hence the effects of a memory transaction’s stores are not visible outside the transaction until the transaction commits; it also means that there are no other conflicting stores by other transactions while runs.

### Get valuables to create a table

The programmer can use it to save valuables in a table. The table will save value of valuable, initial a version number and lock.

### Start Transaction

It is a signal of start a transaction. Let the program know the transaction start.

### Read

Copy value from the table what is created by get valuables function. And also can get current version number and lock.

### Write

Rewrite the new value that has been calculated instead of old value of valuable in table, also can change version number and lock.

### Commit transaction

When the transaction need commit, the transaction will lock the valuable and check version and then update version and valuable.

Next is showing the functions communicates, and they how to work together.



Figure 5 showing transaction how to work

## Procedural Design

In this section, I will design my project how to be coding. Following will show algorithm of my project. And my ideas for my project design.

### Algorithm

* **Create transaction**

When users create transaction, the users can invoke below functions.

createTransaction={} --new class

* **getValuable**

create a table to save the valuable.

function getValuable(a)

function getValuable(a[])

* **STMStart**

function STMStart(a) --create a thread to start a transaction, a equal how many threads are created.

* **Read**

read the value from table, and get the copy value.

Function read(a[]) --get elements from table

Example:

read(a[1][i]) --get value from valuable

read(a[2]) --get current versionNo.

read(a[3]) --get lock

* **Write**

Function write(a[]) --update elements into table

Example:

write (a[1][i]) -- update value of valuable

write (a[2]) -- update current versionNo.

write (a[3]) -- update lock

* **commit**

function commit(a[])

{

 If lock==false

 write(a[3]) --update lock to true, do not let others use the valuables

 If versionNo==currentVersion

 write(a[1][i]) --update new value

 write(a[2]) --update new versionNo.

 Else

 Goto STMStart() --restart

 Else

 Goto commit() --waitting the valuables are available

}

Following is showing users how to use the library in the users’ code. But this only is my idea in my mind. The architecture and how to work you can see figure 5.

t=createTransaction()

a[]=multi valuable --example x,y,z

table m=t.getValuable(a[]) --you can see the Figure 2

or table m=t.getValuable(valuable)

t.STMStart()

a=t.read(m[1][i]) //get value a,b,c

versionNo=t.read(m[2])

a=b+c

c=b-a

b=c-1

t.commit()

end

# High Level Design

In this section, I will focus on some diagrams in low level design view, such as the use case diagram, use cases and system sequence diagram.

1.
2.
3.

## ****Use case diagram****



Figure 6 use case diagram

## Use cases

**Use Case:** Import

**Actors:** Users

**Description:**

the case begin when users want to create transaction. The users type require “name of library” and then can invoke functions to create transaction to process data.

**Use Case:** Create transaction

**Actors:** Users

**Description:**

the case begin when user want to create a transaction to control concurrency. The users invoke create transaction function to create a transaction.

**Use Case:** save valuable

**Actors:** Users

**Description:**

The case begin after users create transaction. Users need invoke a function to create a table to save valuables.

**Use Case:** copy value

**Actors:** Users

**Description:**

The case when the transaction starts, users need invoke read function to copy value of valuable and get current version number.

**Use Case:** commit

**Actors:** Users

**Description:**

The case begin when process finish, users invoke commit function to check valuable is available or not and check the version, to update new value to valuable.

## System sequence diagram



Figure 7 system sequence diagram

# ****Application Testing Scheme****

Testing is an important process of project development. The project is successful or not is depend on the testing. The purpose of testing is to make sure every function modules are working properly as expected, such as:

* It is created transaction correctly or not.
* Check the functions that are written by my own are running well or not. And it is working as my expected.
* Check the lock it is work for avoiding other use same valuable at the same time.
* If the valuable is not available, waiting it available. Or fail to update, need restart transaction.
* If multi threads to run transaction, it is get right value for valuable or not.
* The multi threads and multi transactions are running correctly or not.

I designed a testing scheme to achieve these tasks:

* New a coding in Lua and require library, create transaction. If I can invoke the functions come from library. That means I correctly new a transaction class.
* Try every function to run, if I get the value as I expected, it is running well.
* Create multi threads to run a transaction. At last the value of valuable is right means the lock is work well. For example: x=x+1 (x=1)

Thread1 return 2

Thread2 return 3

Thread3 return 4

Thread4 return 5 and so on.

* Create multi threads run a transaction, create a thread run multi transactions, create multi threads run multi transaction to test my project.

# Conclusion

The design manual should be done before coding. Following this design document, coding will be easier and get clearly how to develop the project and the testing, debugging will be easier to perform in the future.

It focuses on how to implement the feature in the specification. During the design, I get a clear view of the relations and interaction between main functional modules of my project. The ideas of testing are also got during the design.

During the process of application development, some ideas may be changed in coding. During developing, maybe I will get better ways to develop project or testing. If there is any change, I will mention it in the documents coming later.

# Reference

[1]Adl-Tabatabai, A.-R., Kozyrakis, C., & Saha, B. (December 28, 2006). *UNLOCKING CONCURRENCY.*